Integrating automated acceptance testing with requirements engineering

Timo Koskiahde

School of Electrical Engineering

Thesis submitted for examination for the degree of Master of Science in Technology.

Espoo 21.11.2016

Thesis supervisors:

Prof. Marjo Kauppinen

Thesis advisor:

M.Sc. Marko Klemetti

Aalto University
School of Electrical
Engineering

AALTO UNIVERSITY
SCHOOL OF ELECTRICAL ENGINEERING

ABSTRACT OF THE
MASTER'S THESIS

Author: Timo Koskiahde

Title: Integrating automated acceptance testing with requirements engineering

Date: 21.11.2016 Language: English Number of pages: 7+42

Department of Computer Science

Professorship: T3003 Software Engineering and Business

Supervisor: Prof. Marjo Kauppinen

Advisor: M.Sc. Marko Klemetti

Acceptance testing and requirements engineering are two areas of the software development process that support each other, and there has been increasing interest for stronger integration of these two areas. The research problem this thesis aims to answer is: *how can automated acceptance testing be integrated with requirements engineering in a beneficial manner?* A qualitative approach was taken for studying the research problem. Two software development projects in the case company were chosen for the empirical study, and a focus group interview was held for each project. Participants for the interviews were chosen with the purposive sampling method amongst personnel who had worked on the studied projects. Analysis of the collected data was done in two phases: the results for the first half of the interviews were analyzed individually for both projects, while the results for the latter half from both projects were aggregated together for analysis.

The empirical study results suggested two important benefits that could be acquired by integrating automated acceptance testing with requirements engineering: creating a shared understanding of the developed software between the customer and the technical team; and providing more accurate information about the status of requirements. The results also suggested that the most significant challenges in integrating automated acceptance testing with requirements engineering are: convincing customers of the benefits of the practice, and lack of cohesion in test automation practices. A number of good practices for integrating automated acceptance tests with requirements engineering were suggested in the focus group interviews. Two of the most important suggested practices were: increasing the customer's involvement in the process of acceptance testing, and using tags to link acceptance test cases to requirements. Currently, no tools for easy utilization of tag-based linking exist. In order to be able to further explore the use of this practice, such a tool should be developed.

Keywords: software testing, requirements engineering, automated acceptance testing, acceptance testing, traceability, software engineering

AALTO-YLIOPISTO
SÄHKÖTEKNIIKAN KORKEAKOULU

DIPLOMITYÖN
TIIVISTELMÄ

Tekijä: Timo Koskiahde

Työn nimi: Automaattisen hyväksyntätestauksen liittäminen vaatimusten määrittelyyn ja hallintaan

Päivämäärä: 21.11.2016 Kieli: Englanti Sivumäärä: 7+42

Tietotekniikan laitos

Professuuri: T3003 Ohjelmistotuotanto ja -liiketoiminta

Työn valvoja: Prof. Marjo Kauppinen

Työn ohjaaja: DI Marko Klemetti

Hyväksyntätestaus ja vaatimusten määrittely ja hallinta ovat kaksi toisiaan tukevaa ohjelmistokehityksen osa-aluetta, joiden yhteenliittäminen on lähiaikoina herättänyt paljon mielenkiintoa. Tämä diplomityö pyrkii vastaamaan seuraavaan tutkimusongelmaan: *miten automaattinen hyväksyntätestaus voidaan liittää vaatimusten määrittelyyn ja hallintaan hyödyllisellä tavalla?* Tutkimusongelmaa lähestyttiin kvalitatiivisesta näkökulmasta. Tarkasteltavasta yrityksestä valittiin kaksi ohjelmistokehitysprojektia, ja kummastakin projektista järjestettiin fokusryhmähaastattelu. Haastattelujen osanottajat valittiin harkintanäytteenottoa käyttäen projekteihin osallistuneiden henkilöiden joukosta. Kerätyn tiedon analyysi suoritettiin kahdessa vaiheessa: haastattelujen ensimmäisen puoliskon tulokset analysoitiin kummallekin tarkasteltavalle projektille erikseen, kun taas haastattelujen jälkimmäisen puoliskon tulokset molemmista projekteista analysoitiin kokonaisuutena.
Empiirisen tutkimuksen tulokset viittasivat kahteen etuun, jotka voitaisiin saavuttaa liittämällä automaattinen hyväksyntätestaus vaatimusten määrittelyyn ja hallintaan: yhteisymmärrys kehitetyn sovelluksen ominaisuuksista asiakkaan ja teknisen tiimin välillä, sekä tarkemman tiedon saaminen vaatimusten tilasta. Tulosten perusteella suurimmat ongelmat automaattisen hyväksyntätestaamisen liittämisessä vaatimusten määrittelyyn ja hallintaan ovat: käytännöstä saatavien hyötyjen perusteleminen asiakkaille sekä yhtenäisyyden puute testiautomaatiokäytännöissä. Haastattelujen tuloksena tuli myös suosituksia hyvistä käytännöistä automaattisen hyväksyntätestauksen liittämiseksi vaatimusten määrittelyyn ja hallintaan. Näistä kaksi tärkeintä olivat: asiakkaan aktiivisempi osallistuminen hyväksyntätestaukseen ja hyväksyntätestitapausten linkittäminen vaatimuksiin tunnisteiden avulla. Helppokäyttöisiä työkaluja vaatimusten ja hyväksyntätestitapausten linkittämiseen ei tällä hetkellä ole olemassa. Tällainen työkalu tulisi kehittää, jotta tämän käytännön hyödyntämistä voitaisiin jatkossa tutkia lisää.

Avainsanat: ohjelmistotestaus, vaatimustenhallinta, vaatimusten määrittely, automaattinen hyväksyntätestaus, hyväksyntätestaus, jäljitettävyys, ohjelmistotuotanto

Acknowledgements

I'd like to thank the following people for supporting me in writing this thesis:

- My fiancée, Nita Miinalainen, for supporting me through the whole writing process
- Prof. Marjo Kauppinen, for providing tons of feedback and guidance on the thesis work
- My coworkers at Eficode, for providing peer support (and peer pressure!) while working on the thesis
- Everyone who helped with the proofreading of this thesis

Otaniemi, 21.11.2016

Timo Koskiahde

Contents

Terms and abbreviations

Abbreviations

AAT	Automated Acceptance Testing
ATDD	Acceptance Test Driven Development
CSS	Cascading Style Sheets
CI	Continuous Integration
RE	Requirements Engineering
SUT	System Under Test
TDD	Test-Driven Development
UI	User Interface
VV	Verification and Validation

Terms

Devops	A software development methodology
Jenkins	Open source automation server software, used mostly for CI
Jira	A proprietary issue tracking software developed by Atlassian
Robot framework	A test automation framework for acceptance testing
Vagrant	Software for building virtual development environments

1 Introduction

1.1 Motivation

Acceptance testing aims to ensure that the tested software satisfies the acceptance criteria set by its users – and by extension, that the software meets the needs of its users [1]. This means that acceptance testing and requirements engineering are two areas of the software development process that support each other [2][3]. However, acceptance tests tend to be more or less disconnected from the requirements engineering process, that is, the link between requirements engineering and acceptance testing is often missing [4]. There has been increasing interest in the field of software development for stronger integration of these two areas. Especially traceability between acceptance test cases and requirements would benefit from better tool and practice support [4].

This thesis work aims to examine good practices for integrating automated acceptance testing with requirements engineering, and to assess the possible benefits and challenges in using these practices in software development work. The expected benefits of this integration include e.g. receiving more accurate information about the status of requirements via acceptance test cases, and improved communication between customer and technical team.

1.2 Research problem and questions

The research problem this thesis aims to answer is: *how can automated acceptance testing be integrated with requirements engineering in a beneficial manner?*

- RQ1: What kind of benefits can be achieved by integrating automated acceptance testing with requirements engineering?
- RQ2: What kind of challenges might be encountered when integrating automated acceptance testing with requirements engineering?
- RQ3: What are good practices for integrating automated acceptance testing with requirements engineering?

1.3 Scope and focus of this thesis

- This thesis focuses only on software development done in agile contexts. This also means that an agile requirements engineering approach was used in the studied projects.
- A ”pure ATDD (Acceptance Test Driven Development) approach” to requirements management would be to use acceptance test cases as the sole documentation for requirements. However, the reality in the software development industry is that at least currently, most customers are not interested in adopting this kind of approach. A separate requirements management system is often needed for other reasons than simply tracking the status of development tasks,

e.g. for invoicing purposes. Thus, this approach was ruled out of the scope of this thesis.

2 Research method of the empirical study

This section describes how the empirical study for this thesis was planned and executed. First, there is a short introduction of Eficode, the company that cooperated in the creation of this thesis by providing the initial topic and the target projects for the empirical study. After that, the projects that were studied in the empirical study are introduced. This is followed by a description of the research process, and finally, a description of how the data collection and analysis were performed.

2.1 Case description

Eficode is a Finnish software company that was founded in 2005. The company currently employs over 130 professionals in Helsinki, Tampere, Beijing, and Copenhagen. The company focuses on developing intelligent digital services for its customers by utilizing Devops methodologies, and is among the pioneers of Devops and use of big data in Finland.

2.1.1 Project 1

The customer for this project was a company focusing on transportation of raw materials. The produced software is an order and billing management system, used for managing the transportation orders and keeping track of their statuses. The development of the software started in fall 2014, and continued until the fall of 2015. After that, the project entered the maintenance phase of its lifecycle. Follow-up development on the new features for the software was started in spring of 2016, and was currently ongoing at the time of writing.

2.1.2 Project 2

The customer for this project was a company that provides risk management, corporate and sales and marketing information for its customers. The produced software is used for visualizing payment method data. By using the software, a company can see how their customers' payment method usage compares to payment method usage in the whole country. The software was developed in two stages: a prototype version of the software was developed between January 2016 and March 2016, and the first 'complete' version of the software was developed between March 2016 and June 2016. The Scrum framework was utilized during the project.

2.2 Research process

The research problem set in the introduction section is a qualitative one, as the point of interest is in 'how' the integration of these two areas of software development might be done. Several alternatives for the research method were considered, and literature on qualitative research methods was studied in order to decide which method would be best suited for studying the research problem.

Kontio et al. [5] describe that the focus group method is a qualitative research method that is well suited for – among other things – obtaining feedback on new concepts, generating ideas and collecting information on potential problems. Daneva also specifically mentions that the focus group method is well suited for studying phenomena related to requirements engineering [6]. Thus, according to literature on the topic, the focus group method should be well suited for examining the research problem of this thesis, and was chosen as the method for the empirical study.

The high-level process for planning and executing the focus groups was based on the steps for focus group research as described by Kontio et al. [5]. A diagram depicting the research process is presented in figure 1. The first step, defining the research problem, is meant to ensure that focus group research is suited for studying the research problem. This step consisted of studying literature in order to determine which research method to apply to the research problem of this thesis.

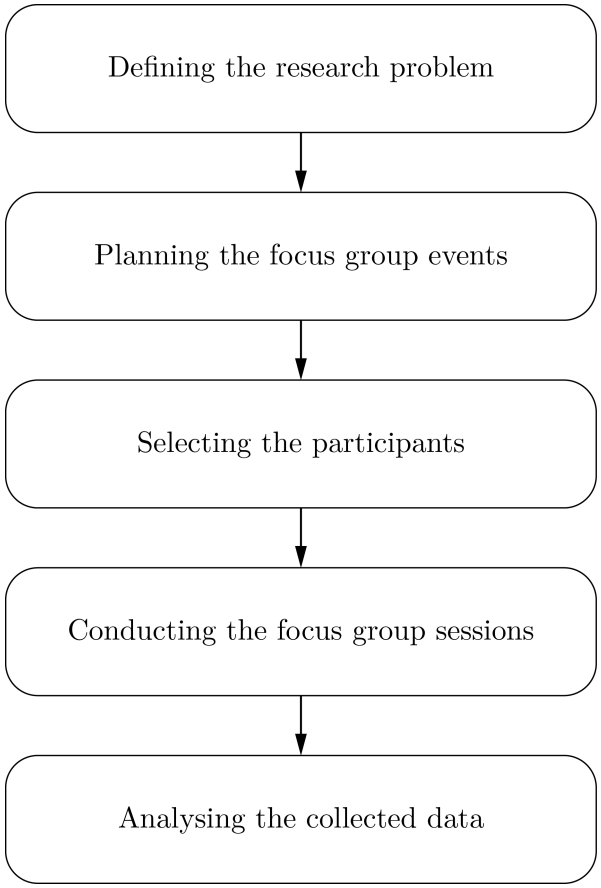

Figure 1: Research process of the empirical study (adopted from [5])

After that, the next step was to plan the focus group events. Krueger and Casey [7] emphasise that the amount of focus group events to be held is an important consideration in any study that utilizes focus group interviews. In order to enable the comparison of results between groups, having more than one focus group is important when using the focus group method. Krueger and Casey presented a rule-of-thumb recommendation that at least three focus group events should be held. However, they

also admitted that planning a focus group study is always a balancing act between the optimal amount of focus groups and the resources available for the study. As the resources and timeframe for performing the focus group interviews for this study were limited, two focus group sessions – instead of the more optimal three sessions – were planned and held for the empirical study. Each focus group session focused on one of target projects. The planned duration for each focus group was two hours, with some leeway included to avoid running out of time in case the discussions would take longer than expected.

The following step in the research process was the selection of focus group participants. Kontio et al. [5] state that usually the amount of participants for each session is between 3 and 12. They also mention that participants for focus groups are usually selected via purposive sampling. Purposive sampling is a non-statistic method of choosing participants where the person organizing the interviews uses their judgement to choose participants from a pool of possible candidates, based on their individual characteristics as related to the session topic.

In this study, the criterion for purposive sampling was that the participants for each group session would provide a representative sample of different roles associated with the target project. The people chosen for the focus group sessions were Eficode personnel who had worked on one of the target projects and had been involved in development, testing or requirements engineering tasks. Unfortunately, including participants from the customer's side in the interviews was not possible due to scope and schedule restrictions of this study.

Table 1 lists the attending participants for the first focus group session, alongside with their roles in the project and the amount of time they had spent on working on the project. The first focus group session focused on project 1, and thus all of the participants were people who had worked on project 1. The first focus group had four interviewees attending out of the five that were invited. Table 2 lists the attending participants for the second focus group session, which focused on project 2. The second group had five interviewees attending out of the six that were invited. Due to scheduling issues, participant B5 joined the second focus group at about the halfway mark, and thus did not contribute answers to all of the questions.

Table 1: Participants of focus group session 1 and their roles in project 1

ID	**Roles in the project**	**Amount of time worked with the project**
A1	Developer	About one year
A2	Developer	Some part-time work in the period of about one year
A3	Writing Robot Framework tests	A few days
A4	Developer	About one and a half months, part-time

Table 2: Participants of focus group session 2 and their roles in project 2

ID	Roles in the project	Amount of time worked with the project
B1	Scrum master and project manager	Since the beginning of the project
B2	Graphical and UI design, front-end implementation	Since the beginning of the project
B3	Graphical and UI design, front-end implementation	Started at the beginning of the project, left at about the halfway mark
B4	Writing automated tests and acceptance tests	Since the beginning of the project, part-time
B5	Back-end development	Since the beginning of the project

2.2.1 Data collection and analysis

The next step in the research process was conducting the focus group sessions. As mentioned earlier, two focus group interviews were held, with each one focusing on one of the target projects. The main goal of the focus group sessions was to gain insight into the research questions set in the introduction section; that is, what kind of benefits, challenges and good practices are related to the integration of automated acceptance testing with requirements engineering. Kontio et al. describe that they designed their focus group sessions to follow a semi-structured format. That is, they had defined the question areas carefully, but not all individual questions were designed in detail. [5] This kind of structure places emphasis on the discussions between the focus group participants. The same semi-structured design was applied for the focus group sessions of this empirical study.

The guidelines for designing a good questioning route, as presented by Krueger and Casey, were utilized in designing the questioning route. The questions started on a general level, and then progressed towards more specific questions. The idea was to help the participants to start talking and thinking about the topics of the study, before progressing onto the actual key questions of the study. [7] With this in mind, the question areas for the focus group sessions – from start to finish – were the following:

- Current status of test automation practices in the project
- Requirements management and customer's role
- RQ3 - What are good practices for integrating automated acceptance testing with requirements engineering?
- RQ1 - What kind of benefits can be achieved by integrating automated acceptance testing with requirements engineering?
- RQ2 - What kind of challenges might be encountered when integrating automated acceptance testing with requirements engineering?

A more detailed version of the focus group interview script (in Finnish) is included as appendix A of this thesis. The research questions were intentionally presented 'out of order', with RQ3 before the others, in order to improve the flow of the questioning route.

Multiple data collection methods were utilized in the focus group sessions. The focus group sessions were recorded with a microphone, and then transcribed in their entirety in order to make data analysis easier. In several of the questions the participants were asked to write down their opinions on the matter on post-it notes. Answers on the notes were then discussed together in the groups, and the notes were also utilized in analysis of the focus group results.

Analysis of the collected data was done in two phases. First, the results for the "current status of test automation practices" and "requirements management and customer's role" question areas were analyzed individually for both of the target projects. Then, results for each question area related to the research questions were analyzed. For each of the research questions, results from both target projects were combined together, and the resulting aggregated data was then analyzed. The structure of the focus group interviews was designed with analysis in mind, as recommended by Krueger and Casey [7]. This made analysis of the results a relatively straightforward task, as the format of the focus group interviews correlated with the format in which the results were presented.

3 Literature review

This section makes an overview of literature related to this thesis' topic. First, a definition for software testing and its goals is established. Then, the areas of automated acceptance testing and requirements engineering are explored. Finally, the integration of automated acceptance testing with requirements engineering is examined.

3.1 Definition and goals of software testing

The Guide to Software Engineering Body of Knowledge (SWEBOK) defines software testing as follows: "Software testing consists of the dynamic verification that a program provides expected behaviors on a finite set of test cases, suitably selected from the usually infinite execution domain." That is, testing means executing the SUT (System Under Test) with a certain set of inputs in order to evaluate the system's behavior. [8] A common misconception is that this would mean the aim of software testing is to show that a program works correctly. Myers et. al argue that a better definition for software testing is the execution of a certain program with the intent of finding errors in it. [9] Both definitions mention that software testing involves *executing* program code of the SUT. This is an important aspect of software testing, and it separates software testing from other software quality management techniques, such as static code analysis [8].

Ammann et al. explain that the concepts of software *error*, software *fault*, and software *failure* allow one to distinguish software testing from debugging. A software fault is a static defect in the software. A software error is an incorrect internal state of the software, and the manifestation of a fault. A software failure is an externally perceivable, incorrect behavior of the software in respect to its requirements or its expected behavior. Thus, software debugging is the process of identifying the underlying fault (and by extension, identifying the error that manifested as the fault) in software code when a failure has been detected. Testing is the process of executing the software code in order to evaluate its behavior, so that failures in the software can be found. [2]

As software testing is a quality assurance activity, the fundamental goal of software testing is to improve software quality. In order to achieve the rather vague goal of improving quality, however, some lower-level goals must be defined first. Ammann et al. emphasise that the two most important goals for software testing are *verification* and *validation*. [2] Verification attempts to ensure that the software is built correctly: that the software works properly and that it corresponds to its requirements specification. Validation refers to the process of ensuring that the software meets the needs of its users. In other words, that the right software has been built. In order to be able to successfully test the software for validation or verification, the user needs and requirements specification must be known well enough. That is, one should always be able to decide if the results of a testing run are acceptable or not; otherwise the testing effort is useless. [8]

From an economical perspective, improving quality means improving the value

provided to the users of the software. By finding and removing failures in the software during development, the software is made more reliable. This should (at least in theory) translate to higher value to users of the software. [9] Another way to view software testing as a way to manage risk related to possible defects in the software [8]. As such, software testing always involves trade-offs between test coverage and resources available for the testing effort. Catching faults earlier in the development cycle reduces the cost of fixing said faults [2].

The definition of software testing has also changed over time. Instead of a waterfall-like approach where testing is a 'phase', the view of software testing has changed into a more constructive and preventive one. Planning for software testing should start alongside with requirements engineering activities, and testing should be performed and developed alongside the software that is being developed. [8] Ammann et al. emphasise that even though tests cannot be executed early during development, the process of defining tests can help identify a significant portion of faults in requirements and design. [2]

3.2 Levels of testing

Software tests can be written based on different artifacts, such as requirements and specification documents, design documents or source code. Software testing activities can be divided into levels based on the software development activity it accompanies. Information for each test level can be derived from the associated software development activity. Figure 2 depicts the so-called 'V-model' of testing levels. [2]

Ammann et. al describe five different levels of software testing and the software development activities that they assess in the following fashion [2]:

- Acceptance testing – assess software with respect to requirements
- System testing – assess software with respect to architectural design
- Integration testing – assess software with respect to subsystem design
- Module testing – assess software with respect to detailed design
- Unit testing – assess software with respect to implementation

Note that the levels of V-model do not necessarily imply phases when the testing activity is applied, but rather the types of faults the activity aims to detect. Even though the V-model might have originally been associated with a waterfall-like process, it can just as well be applied in an agile development context. The software development activities that are mentioned in V-model do take place also in agile contexts, but not necessarily in the same form they would be in a waterfall-like process. It is recommended to design the tests corresponding to each software development activity alongside the development activity itself. For example, design of acceptance tests should be performed alongside requirements analysis, even though the tests cannot be executed successfully yet at that point of time. [2]

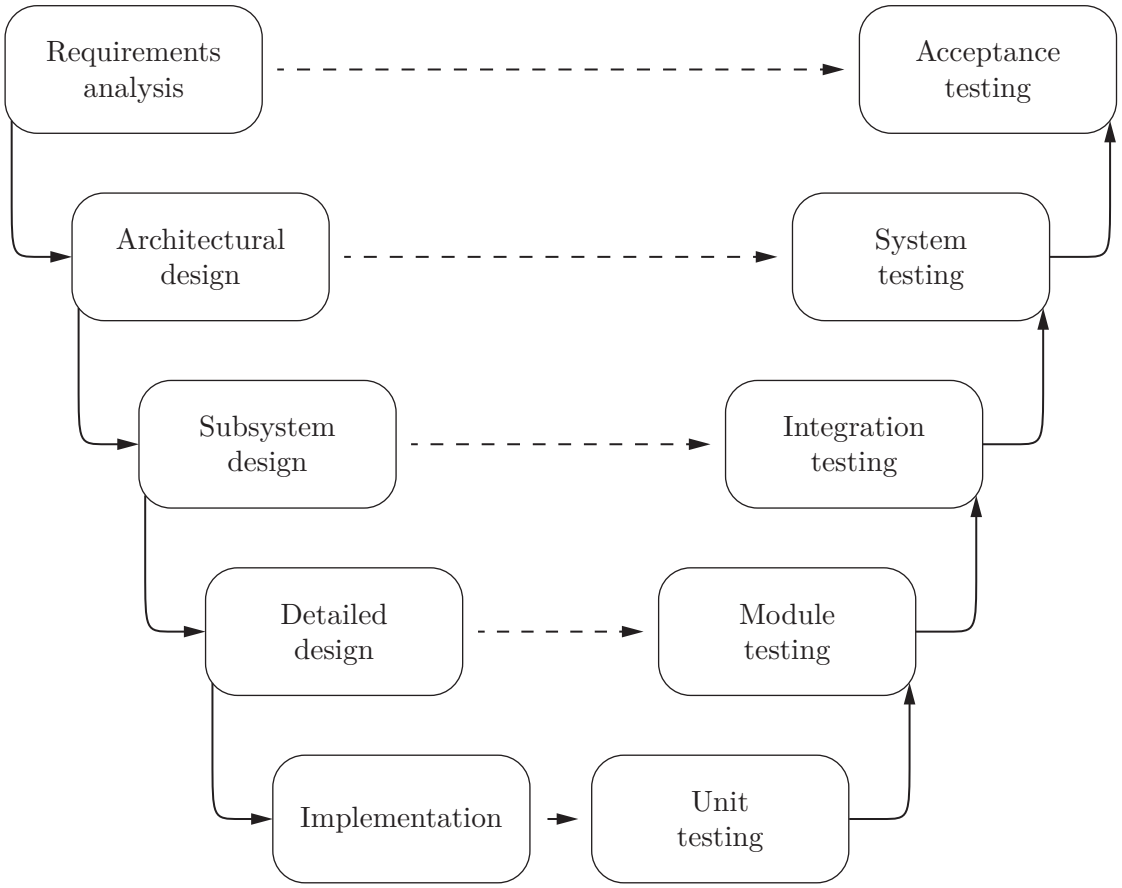

Figure 2: V-model of software testing levels (adopted from [2])

The focus of this thesis work is on the highest level of the V-model, requirements analysis and acceptance testing. The goal of requirements analysis is to capture the customer's needs, while acceptance testing aims to ensure that the software meets the customer's needs. Thus, acceptance testing tests the software for validation. [2]

3.3 Automated acceptance testing

Acceptance testing aims to ensure that the tested software satisfies the acceptance criteria set by its users – and by extension, that the software meets the needs of its users. However, manual acceptance testing is a tedious, repetitive, time consuming – and thus, expensive – activity. Additionally, manually running all acceptance tests can easily form a major bottleneck for testing, and create a major overhead for each product delivery. Automating the process of acceptance testing is usually a lucrative prospect, and agile methodologies often consider automated acceptance tests a must-have. [1] [10]

The basic idea of AAT (Automated Acceptance Testing) is to document the requirements in a format that can be easily and repeatedly tested [10]. Haugset et al. state that AAT is just as much about communicating requirements as it

is about testing and automatic verification. AAT facilitates the communication of requirements between developers and customers, and can help create a shared understanding of the requirements. Well-maintained automated acceptance test cases also act as up-to-date documentation for the requirements. As a result, AAT can also be considered a merge between documentation-centric RE and software-focused agile RE. [11]

AAT provides many benefits from a testing viewpoint. The most obvious benefit is providing decent test coverage for important parts of the software. Haugset et al. also reported that developers using AAT had more confidence to make alterations to the code, as they could easily detect if code changes broke something. [10] Acceptance tests can also catch many bugs that lower level tests, such as unit tests, are unable to detect [1].

Talby et al. propose the use of AAT for measuring how 'done' the software under development is. They propose that the system is deliverable once all of its acceptance tests pass. The measure of progress depicted by passing acceptance tests may indeed be more accurate than some other progress metrics, such as percentage of code written, time invested, and so on. [1] However, Haugset et al. advise against trusting AAT blindly, as passing automated acceptance tests may provide a false sense of control. If an acceptance test case does not accurately capture the intentions of the corresponding requirements, the passing test result may be very misleading. They also urge that the cost of writing and maintaining automated acceptance tests should be weighed against its potential benefits in advance. Additionally, they feel that AAT has not yet reached maturity, and that research literature has so far addressed many topics related to AAT too weakly. [10]

3.4 Requirements engineering

Requirements engineering refers to the systematic handling of requirements. A software requirement is a property that a software system must have in order to solve some problem in the real world. An essential part of a software requirement is that it must be possible to verify that the finished software satisfies it. Verifying some requirements might be difficult or expensive; e.g. nonfunctional requirements tend to be harder to verify than functional ones. [8]

The way that agile methodologies handle requirements engineering differs from that of 'traditional' methodologies. The traditional approach to RE emphasises heavily the use of documentation and the aim to find out a reasonable set of requirements before starting the actual implementation. This does not translate very well to the dynamic business environment that often acts as the context for software development today. Requirements might be hard to define beforehand and change quickly, so after a certain point, investing more effort into pre-planning the requirements is not worth the effort. According to Käpyaho et al., many researchers have come to the conclusion that continuous changes in requirements are inevitable. [12]

Agile methods have strived to solve many of the problems that the traditional, planning driven RE approaches have with regards to change. In agile methods, design is mostly a continuous activity, and changes to requirements during the project are

assumed. Agile methodologies differ somewhat in how the continuous design and refactoring should be implemented. [12] On a general level, agile requirements engineering emphasises the importance of customer involvement over reliance on formal requirements documentation. The used techniques are not as important as the emphasis on customer involvement throughout the development process. [13] As an example of an agile requirements engineering approach, Bjarnason et al. [14] describe that in a case project that they studied, user stories were utilized for documenting requirements. User stories were utilized because the project participants felt that user stories facilitate communication between business and technical roles. That is, the communication between project participants was seen as more important than having a formal, detailed documentation of each requirement.

Käpyaho et al. mention that even though agile RE methods emphasise direct communication with customers, communication issues might still occur. The absence of comprehensive documentation about requirements increases the need for communication with customer representatives. The agile RE methods assume that the customer representative is very accessible, and that they can participate daily in project activities. In many projects, this is not a realistic expectation. If the customer representative lacks the resources to participate enough in RE activities, the lack of high-quality communication can lead to inadequate or even incorrect requirements. [12]

3.5 Linking testing and requirements engineering

Requirements engineering (RE) and verification and validation (VV) both aim to support software development in order to ensure that the finalized product will satisfy customers' expectations regarding functionality and quality. In order to achieve this, RE and VV have to be aligned and their activities organised so that they fit well together. Well-aligned RE and VV can effectively support development activities between the initial definition of requirements and acceptance testing of the software. On the other hand, weak alignment of RE and VV can lead to a multitude of problems, such as inefficient development and deficiencies in functionality and quality of the produced software. [3]

Bjarnason et al. [3] mention that traceability has often been the focal point when alignment of RE and VV has been discussed. Traceability means implementing some sort of permanent and visible reference between artifacts, such as requirements and their corresponding test cases, so that the connection between the artifacts can be traced in either direction (i.e. from test case to requirement(s) and vice versa) [15]. Traceability mainly focuses on linking documents or artifacts together. Another aspect of the alignment of RE and VV, linking of people together, is more rarely discussed in research. [3]

Uusitalo et al. [4] reported that in their study, many benefits were gained from the traceability of artifacts. Test coverage improved, as traceability made more apparent which requirements were covered by test cases and which were not. Change management was made more efficient, as the effects of requirements changes could easily be traced into the test cases associated with the change. An increase in the

efficiency of error removal was also reported, as testing personnel could more easily change their viewpoint between tests and requirements.

Uusitalo et al. [4] also reported many challenges that hinder the usage of traceability in their study. Traceability between requirements and tests was rarely maintained in practice. This was caused primarily because of failure to update traces when requirements changed due to schedule and budget reasons. Inadequate tool support for traceability also hindered the maintenance of traces. Low quality of requirements was also seen as a hindrance to maintaining traces, as there were no quarantees on how well the requirements actually covered the functionality of the software.

In addition to traceability, which links together artifacts and documents, alignment of RE and VV can also be performed by linking people together. Both practices share the common goal of providing testing personnel with authoritative and up-to-date information about requirements. Uusitalo et al. [4] found in their case study that participants usually found the linking of people together almost as important as the linking (or tracing) of artifacts together. In addition, the links between documents and people were not seen as redundant to each other; instead, both links supported each other when applied together. The findings of the study also emphasised the importance of the testing personnel to achieve comprehensive knowledge of the application domain. This assists the testing personnel in understanding the requirements better, and helps in the design of test cases.

4 Empirical study

This section covers the results of the focus group interviews. First, a set of good test automation practices is introduced. Then, the current status of test automation practices in the target projects is examined. After that, the results concerning the three research questions that were set in the introduction section are examined. The first research question, RQ1, provides motivation for the integration of automated acceptance testing and requirements engineering. Once that is established, RQ3 is examined and good practices for integrating these two areas of software development are discussed. Finally, RQ2 is examined in order to discuss challenges that might hinder the integration.

4.1 Test automation best practices

The Eficode test automation guide [16] is a general-level guide for good test automation practices. The guide was released in early 2016, and thus provides a current view into the state of what are considered good test automation practices. In order to find out the current state of test automation practices at the case company, the test automation guide was chosen as a reference against which the test automation practices of the target projects were compared. The target project case descriptions are presented in section 2.1. The most relevant and condensed part of the test automation guide was the listing of six test automation best practices. Table 3 contains the aforementioned listing of good test automation practices. Out of these six practices, four practices that are the most relevant to the topic of this thesis were chosen. Adherence to these four practices in the target projects was examined in the focus group interviews.

Table 3: Good practices for test automation, as described in the test automation guide [16], and whether they are relevant to the subject of this thesis.

Good practice	**Definition**	**Relevant?**
(GP1) Write test cases together and integrate them with requirements	When test cases are written together, everyone who participates in the development project has to process the requirements and functionalities of the software. Thus, writing test cases together creates a shared understanding of what is trying to be achieved with the software.	Yes
(GP2) Execute tests after every code change	When tests are executed after every code change, possible errors and failures in the software are caught as quickly as possible. This also prevents the building of new functionalities on top of a broken implementation, so that these functionalities don't have to be undone later. This kind of way to operate is called *continuous integration*, and it is one of the cornerstones of Devops methodologies.	Yes
(GP3) Execute tests also for development branches	In order to facilitate smooth merging of multiple development branches into a single release, it is important to execute test cases also for the changes done in development branches.	No
(GP4) Execute tests in parallel	Running tests in parallel enables one to receive feedback faster and to improve the software's architecture. No dependencies between test cases are formed when test cases are run simultaneously, preferably in environments that have been booted up specifically for the testing run.	No
(GP5) Automated test cases are a part of the product	Test cases have to be maintained, just like regular program code. The aim is obviously to have test cases that are as easily maintained as possible. If functionality of the tested software changes, however, test cases or keywords have to be updated accordingly.	Yes
(GP6) Emphasise good test data	Optimally, the testing environment matches the production environment so well that passing test case execution in the testing environment means the software is ready to be released to customer use. An important part of this kind of testing is the test data used in the testing, which defines what kind of information is accessible from the testing environment.	Yes

4.2 Current status of test automation practices in projects

4.2.1 Project 1 - Test automation status

First, the focus group participants were asked to provide a short summary of what kind of test automation practices had been used in the project. Two types of tests

were used in the project: unit tests and acceptance tests. Unit tests were written with the unit testing classes of Ruby on Rails, while acceptance tests were written with Robot Framework.

At the start of the project, one of the developers laid out groundwork for both unit and acceptance tests for the software. The test cases mostly covered the main flows of the software. For most of the earlier part of the project, tests were lagging behind. As time went on, the test cases started becoming outdated.

After the initial development project, but before follow-up development began, a dedicated 'tester' (marked with ID 'A3', see table 1 for participant information) was assigned to work on the project. When she joined the project, she fixed the existing tests and added some new acceptance test cases, thus reducing the amount technical debt in the project's testing. Majority of the Robot Framework test cases in the project were eventually written by her.

'A3' commented that there was no up-to-date requirements document, and that instead, she wrote the Robot Framework test cases based on an exploratory approach. She manually went through important flows of the software, ensured that nothing looked broken at a glance, and then wrote test cases based on her understanding of how that part of the software should work. Thus, testing was mostly done for verification, and not so much for validation (see section 3.1 for definition of these terms).

A Jenkins instance had been set up to run a set of Robot Framework tests whenever a deploy to the staging environment was done. However, as test coverage was low and tests were not entirely up-to-date, acceptance testing of the software was mostly done manually by the customer, instead of utilizing the automated Robot Framework tests. This also meant that occasional test failures were largely ignored.

Overall, the sentiment was that testing had been done somewhat inconsistently and not very thoroughly in the project. More than one participant stated that there weren't any good test automation practices used in this project. Despite this, some developers tried to keep the tests up and running, while others admitted that they usually did not run or update the tests. The participants mentioned multiple obstacles to following good test automation practices in the project, including:

- Customer has been strict with effort estimations, so there hasn't been enough time for good test automation practices
- The software tended to freeze during testing, so testing was unreliable most of the time
- Customer had not allocated enough resources for testing

4.2.2 Project 1 - Requirements management

Requirements for the software were originally defined and written down by project members on Eficode's side, and then approved by the customer. Initially, the requirements were well-organized and clearly defined. However, the customer seemed to be uncertain of what they actually wanted from the software, and changes to the

requirements were requested many times after the developers had started implementing them. This could be repeated many times over for a single feature, with the team having to discard some of their previous work with each change request. This is an indication that the project seemed to lack a shared understanding between the client's representatives and the project team, resulting in unnecessary work.

Jira was used by the development team for managing requirements during the development project. In this project, the customer preferred to communicate change requests via email instead of using Jira. Over time, this resulted in requirements and their change requests becoming scattered over multiple email conversations and Jira tickets. Thus, the developers could not trust that the information in Jira was up-to-date. Eventually it became very hard to track all the information related to a single requirement, making requirements management difficult.

4.2.3 Project 1 - Customer involvement & communication

The customer participated in the original requirements definition process by approving the requirements written by the development team. After that, the customer took on a more passive role in the project. The developers felt that the customer had not allocated enough resources for testing, and that the customer did not have enough time to do acceptance testing properly. Often, the customer would accept a feature as 'working correctly', only to comment later that the feature is not actually working how they would have wanted. This lengthened the feedback cycle between developing a feature and receiving feedback from it.

The developers were frustrated with the situation, as it was hard to know how exactly each feature should work and what kind of corner cases and quirks each feature might have. In order to tackle the issue, the development team had multiple conversations with the customer regarding the problems with acceptance testing, and explained that not doing acceptance testing properly would result in the development team doing unnecessary extra work. Later on, a person from the customer side started doing some acceptance testing on the staging server for new features, so there has been improvement on customer's involvement in testing.

The developers also had issues with the way the customer communicated new requirements to them. Participant A2 described that receiving requests for new requirements was like "being handed a figurative ball of yarn from the customer, and then having to untangle that ball of yarn in order to figure out what the customer had meant with the requirement in question". That is, the development team and the customer seemed to lack a common understanding when new features were discussed. This issue was made worse by the previously mentioned problems with acceptance testing. As the development team did not receive feedback from the customer quickly enough, and they had a hard time understanding what the customer wanted when they communicated new requirements, they were not very confident that the features they had implemented would fulfill the customer's needs.

4.2.4 Project 1 - Test automation good practices

Participants of the focus group were shown a slide show, with each slide containing one of the four good practices from the test automation guide [16] that were deemed relevant for the topic of this thesis. The participants were then asked to write down a post-it note whether or not they felt that that particular good practice had been followed during the project. Table 4 lists the responses from each participant of focus group 1, alongside with some comments brought up during discussion afterwards.

Table 4: Good test automation practices followed in project 1, as described by focus group participants. More thorough explanations of the good test automation practices mentioned here are listed in table 3.

Good practice	**A1**	**A2**	**A3**	**A4**	**Comments**
(GP1) Write test cases together and integrate them with requirements	No	No	No	No	"Tests and requirements haven't been examined together properly." [A2] "Tests were not linked to requirements." [A3] "This was not followed during follow-up development." [A4]
(GP2) Execute tests after every code change	Yes and no	No	Yes	No	"There's hardly any unit tests, but Robot Framework tests are executed when deploying." [A1] "Tests are run about 1/4 of the times." [A2] "Usually not." [A4]
(GP5) Automated test cases are a part of the product	No	No	No	No	"The customer is not interested in tests. In this case, the time spent on developing tests should be included in effort estimates." [A2]
(GP6) Emphasise good test data	Yes	Not sure	Yes	No	"Maybe? The project does have some fixtures defined, but I do not recall having defined them myself." [A2]

Participants unanimously agreed that GP1 had not been followed during the project. Test cases were not written together with the customer, and test cases were not integrated with requirements either.

Participants seemed more divided on whether GP2 had been followed or not. As mentioned earlier in section 4.2.1, a Jenkins instance had been set up at a later stage in the project to run the Robot Framework tests whenever a deploy to the staging environment was done. The person responsible for writing most of the Robot Framework tests, A3, had apparently run the tests systematically after each code change. Most of the other participants commented that they executed the tests only sporadically. The participants cited the lack of unit tests as one of the reasons for this.

Participants also agreed that GP5 had not been followed. Apparently, the test cases were not regularly maintained when other changes to the software were made, and testing was lagging behind for the whole duration of the project. A3 was also doubtful that the Robot Framework tests she wrote to the project at a later time would be maintained by anyone in the long run. One of the developers mentioned that the customer's strict restrictions on estimated effort for features meant that usually there was not enough time for maintaining the test cases or writing new ones

when new features were developed.

As for GP6, opinions on whether test data had been good or not were divided. Some developers had interpreted the question to mean the data used in test fixtures, whereas others had understood this to mean the data in the staging environment's database. The participants agreed that the fixture data is outdated and of bad quality. On the other hand, the data in staging environment's database was initially copied from the production database. The staging environment data was also occasionally updated, e.g. whenever there had been a long break in development or after the current data in staging environment seemed to be outdated. Thus, data in the staging environment database was of good quality and it was up-to-date most of the time.

As a general observation, the lack of consensus on whether or not some of the good practices had been followed implies that the good practice recommendations in the test automation guide [16] are somewhat ambiguous. The ambiguosity of the good practice recommendations was also specifically mentioned by some of the participants during the focus groups.

4.2.5 Project 2 - Test automation status

Like in the first group, the focus group participants were asked to provide a short summary of what kind of test automation practices had been used in the project.

The software was tested on three different levels: unit testing was done with JUnit, acceptance-level testing with Robot Framework, and there were also some integration-level SOAP tests. As mentioned earlier in the case description section 2.1.2, the project was divided into two parts: prototype development and main development project. The prototype development lasted for about two months. After the prototype was complete, the main development project began. At the beginning of the main development project, participant B4 joined the project, and proper testing effort for the software was started. Majority of the testing effort was done during the main development project. During the prototype phase, only a small amount of Robot Framework tests and a Jenkins job executing them upon deploy had been written.

After joining the project, participant B4 took on the role of a dedicated 'tester', while others focused on developing new features. Unlike in project 1, Robot Framework tests were for the most part written simultaneously with the development of new features. The dedicated 'tester' wrote Robot Framework tests as needed for whichever features had been completed at the time. The tester has also maintained the Robot Framework tests together with other members of the development team. Despite the fact that the Robot Framework tests were up-to-date and had good coverage, the client preferred to do manual UI testing for acceptance testing the software. Thus, the Robot Framework tests were more accurately 'acceptance-level' tests instead of actual acceptance tests.

Unit testing should have been the developers' responsibility, but the dedicated 'tester' did end up writing also some of the unit tests at the beginning of the project. Participant B1, who acted as project manager for the project, commented that the developers' unit testing effort was lagging behind a bit at the start of the project. Participant B4 commented that the ratio of acceptance tests and unit tests in the project was supposed to adhere to the 'testing pyramid': unit tests should have been the largest portion of the project's tests, and acceptance testing should have been the smallest. It ended up the other way around, as the amount of acceptance tests was larger than the amount of unit tests. Some of the developers fixed most of the unit tests at a later stage in the project, so the situation did improve eventually.

A Jenkins instance was set up to run the Robot Framework tests after each frontend deploy, and also automatically once a day, regardless of the amount of deploys. The Jenkins instance was also used as an improvised monitoring system during the weekends, as the staging server had a bad habit of crashing occasionally. If the daily Jenkins test run did not pass during the weekend, it meant that the staging server had crashed. The automated test execution which was done once a day was disabled near the end of the development project, as the team decided that there's no point in automatically running the tests if no changes have been made. Otherwise the Jenkins instance was kept updated and running for the whole duration of the project.

4.2.6 Project 2 - Requirements management & customer involvement

A Jira installation provided by the customer was used for managing requirements in the project. The customer also actively used the Jira board. Despite the fact that in Scrum, only the project owner is supposed to create new tickets, participant B1 mentioned that this rule was not strictly adhered to. Thus, the development team was also allowed to create new tickets to Jira as needed.

The customer was described as having been actively involved in the project. As usual, the customer was not that interested in testing itself, but was instead interested in having the software being built properly and with good quality. People from the customer's side participated in defining use case scenarios for the software, and they were also active in prioritizing requirements. They also helped in splitting user stories into smaller pieces, so that they could be realistically be implemented in a single sprint. In addition, people from the customer's side helped in validating the test data used for calculations in the software.

Participant B1, who acted as project manager in the project, gave a lot of praise to the customer for taking on such an active role in the project. The customer has been really serious about the fact that you have to 'choose your battles', i.e. that you have to prioritize things during software development. The development team worked in the customer's premises, and the participants stated that this helped the customer in taking a more active role in the project. B1 also commented that unlike in many other projects, the customer's product owner has had enough time to spend on the project, which has helped the development team and the customer to co-operate more closely.

4.2.7 Project 2 - Test automation good practices

Like in the first focus group, the participants of the second group were asked which of the four good test automation practices that were deemed relevant for the topic of this thesis had been followed in the project. Table 5 lists each participant's responses.

Table 5: Good test automation practices followed in project 2, as described by focus group participants.

Good practice	**B1**	**B2**	**B3**	**B4**	**Comments**
(GP1) Write test cases together and integrate them with requirements	No	No	Yes	No	"The customer was involved in defining use cases, so I thought the question also meant those. That's why I answered 'yes'." [B3] "Customer was not involved in defining test cases." [B1] "Tests were not linked to requirements, but tests were discussed together to some degree." [B4]
(GP2) Execute tests after every code change	Yes	Yes	No	Yes	"Mostly yes, but the client's environment imposed some limitations" [B1] "Tests were not run for small front-end commits." [B3 and B4]
(GP5) Automated test cases are a part of the product	Yes	Yes	Not sure	Yes	"Yes, but I doubt that they will be maintained in the future." [B4]
(GP6) Emphasise good test data	Yes	Yes	Yes	Yes	"We had a database dump of production data ported to the staging server, so the data was basically the same as in production." [B4]

For the most part, the participants agreed that GP1 had not been followed in the project. There was some confusion about the wording of the question, as the customer had been involved in defining use cases, but not test cases. The dedicated 'tester', B4, had discussed with other members of the development team about how to implement the test cases. The cases were not linked to requirements in any formal way.

However, participant B1 stated that some of the user stories had been written in 'test-case-like' manner, that is, their acceptance criteria was written in a format that could be translated into test cases with little effort. B1 explained that this is the first step in one approach to creating automated acceptance tests: first, you write user stories in written form, and then automate them by writing automated acceptance tests. The alternative would be to write the use cases directly into the form of automated acceptance tests. However, as discussed earlier in section 4.2.5, the customer preferred to do acceptance testing via manual UI testing instead of automated acceptance tests. Thus, the written user stories were also needed, as the customer did not want to rely only on automated acceptance tests.

For GP2, participants agreed for the most part that this practice had been followed in the project. Once again, there was some disagreement over the wording of the good practice, but the general sentiment was that tests were usually executed with each

code change. However, there was one notable exception for following this practice. As the Jenkins instance automatically executed tests each time the frontend was built, in situations where multiple commits were deployed at the same time, this would result in the system running the tests repeatedly, once for each commit. In such situations, the team decided it was better to make an exception to the practice, and run the tests only once for the whole set of commits, in order to speed up the frontend development work. The restrictions imposed by the development environment - which was provided by the customer - was also mentioned to have hindered the testing effort to some degree.

As for GP5, the participants agreed that this practice had been followed. The only exception was participant B3, who declined to comment, as he had left the project in an earlier phase, and thus had no knowledge if this had been followed or not. As mentioned earlier in section 4.2.5, writing of the Robot Framework tests had mostly been done simultaneously with the development of new features for the software.

Participant B4, along with some other members of the development team, had taken responsibility for maintaining the Robot Framework test cases during development, and the general sentiment of the participants was that the test cases had been maintained and updated properly during the project. As discussed earlier, the acceptance-level Robot Framework tests had accounted for the majority of the test cases in the project. The situation with unit tests and SOAP tests was not as good as with the Robot Framework tests. Especially the SOAP tests had become outdated during the course of the project.

Though most of the test cases were in a good shape at the end of the development project, the participants had some concerns about how the test cases would be maintained in the future. The development team had not discussed with the customer if anyone would maintain the tests on their end. At the time the development project ended, nobody knew if there were plans for doing additional development on the software in the future. The participants discussed that if someone on the customer's side would do additional development on the software, there is the risk that they might ignore maintaining the tests.

The participants were all in agreement that GP6 had been followed. At an early stage in the project, the development team took a database dump from the customer's production database and ported it to the staging environment for testing purposes. Thus, the data used for testing the software during development should have been very similar to the data that the software would handle when in production use. However, participant B5 – who joined the focus group session after this question had already been discussed – mentioned that there had been quite a bit of problems with the transition from the previously used mock data to using the data in the production database dump. Other attending focus group participants had apparently been unaware of this, as the problems had mostly been visible to the backend developers. Overall, this good practice had been followed for the most part in this project.

4.3 RQ1 - What kind of benefits can be achieved by integrating automated acceptance testing with requirements engineering?

Now that the current situation of test automation practices in the target projects has been established, the potential benefits that could be achieved by integrating automated acceptance testing with requirements engineering are examined. Three categories of potential benefits were identified in the answers provided by participants from both focus groups. Table 6 provides a summary of the answers by focus group participants, ordered by the aforementioned categories.

Table 6: Potential benefits that could be gained by integrating automated acceptance testing with requirements engineering, organized by category.

Category	**Potential benefits**
Shared understanding of the software	• Reduce possibility of having to discard work due to misunderstanding requirements • Detect errors in requirements definition early on • Technical team can more easily ensure that the software provides the expected value to the customer
Better visibility into status of requirements	• Provide more accurate information about requirements' implementation status to non-technical project participants • Encourage non-technical project participants to be more involved in the development
Increased noticeability for automated tests	• Raise developers' awareness of automated tests and their status • Reduce possibility that automated tests are not maintained

4.3.1 Shared understanding of the software

One of the most often mentioned possible benefits in the focus groups was the aspect of creating a shared understanding of the software being developed between the customer and the technical team. The description of GP1 from the test automation guide [16] also mentions this benefit. The aim in involving the customer in the process of defining acceptance tests is to make both the customer's representatives and members of the technical team to think about the requirements together.

By involving the customer in defining acceptance tests, the technical team basically demonstrates to the customer how they have understood the requirements. Thus,

the customer can detect if the technical team has misunderstood something about the requirements, and give immediate feedback so that the misunderstanding can be rectified. This can help reduce the possibility of the technical team having to discard a part of their work on the software due to having misunderstood the requirements. As mentioned earlier in section 4.2.2, the development team in project 1 occasionally struggled with this exact problem. They felt that having a shared understanding of the software with the customer would help in preventing such problems in the future. On the other hand, by having the customer think more about the practical aspects involved with implementing the requirements, the customer might also discover faults in the requirements that they have initially defined. Ideally, a shared understanding of the software that is being developed should ensure that the technical team has understood the requirements correctly, and that the customer is getting what they want from the software.

Another facet of this benefit is the value provided by the software to the customer. Some of the focus group participants felt that they usually did not know what are the important functionalities and features in the software. If the technical team understood better what is the value that the software provides to the customer - that is, why the software is being developed in the first place - they could better prioritize both the development and testing efforts on the most important parts of the software. Participant B1 elaborated on this, and stated that it is not actually in the customer's best interests to have the technical team just develop the software according to the specification, without asking any questions. Instead, part of the value that the technical team provides to the customer is that the team uses their own brains to think about the best way to accomplish what the customer really wants from the software. This additional value provided by the technical team is lost if the developers do not understand what is the value that the software would provide to the customer.

4.3.2 Better visibility into status of requirements

Better visibility into the status of implementation for requirements was mentioned as a possible benefit by the focus group participants. Participant B1 explained that in general, whenever you define a requirement for software, you should also have a set of (preferably automated) test cases for testing that requirement. Integrating acceptance test cases to the requirements, so that you could easily see which tests are linked to a particular requirement, and also the status of those tests (i.e. do the tests pass or not), could enable better visibility into the actual status of those requirements. This would be especially helpful for non-technical persons, e.g. business people from the customer's side. As business people are the ones that usually handle the funding related to software projects, having a better visibility into the status of requirements would provide them with more accurate information about progress on the software's development. That is, what are they receiving in exchange for the money they have spent on developing the software. This also ties heavily into the aforementioned aspect of creating a shared understanding of the software between the customer and technical team. By providing more accurate information about the development to

non-technical people, they might be more willing to e.g. define the acceptance tests together with the technical team.

As it stands, it is not usually possible for non-technical people to make the connection between requirements and their associated test cases. As the focus group participants described it, usually you can only see from a continuous integration tool (e.g. Jenkins) that the tests either did or did not pass as a whole. Finding out anything beyond that requires at least a moderate amount of technical skill. Thus, non-technical people involved in the project, such as business people from the customer's side, have very little information at their disposal in the case that the automated tests do not pass. The focus group participants argued that if you could easily see from a project management tool (e.g. Jira) which requirements the non-passing tests were associated with, the non-technical personnel would have a much better picture of the situation. That is, which features or parts of the software are affected by the problem.

The focus group participants also mentioned that in order to provide better visibility into the status of requirements, the project team has to build and maintain a set of automated regression tests. Thus, integrating acceptance tests with requirements engineering encourages the project team to adhere to good test automation practices, such as GP5 from the test automation guide [16].

4.3.3 Increased noticeability for automated tests

The overall sentiment of the focus group participants was that currently, automated test cases are somewhat disjointed from other parts of a software project. That is, automated tests usually have a rather low profile, and so they tend to go unnoticed by developers. As a result, developers often tend to neglect maintaining automated tests. Participant B1 stated that this seems to be a recurring problem in projects where automated tests are utilized. The focus group participants' answers indicate that by integrating automated tests more tightly with requirements, one could raise the developers' awareness of the automated tests and their status. This could then lessen the possibility of the developers neglecting maintenance of the tests.

Participant B5 also pointed out how it makes little sense that automated testing is so disjointed from other parts of a software project. He mentioned as an example that some developers already add the corresponding ticket number from the requirements management system to each code commit that they do. This links together code commits and tickets from the requirement management system, allowing for easy traceability between the two. In a similar fashion, it would make sense to integrate automated tests to requirements. This could make the maintenance and development of the automated tests easier, by making them a more prominently visible part of the software project.

4.4 RQ3 - What are good practices for integrating automated acceptance testing with requirements engineering?

The participants in both focus groups were asked to write on post-it notes some suggestions on good practices for integrating automated acceptance testing with requirements engineering. Five categories of suggested practices and improvements were identified from the answers on post-it notes and discussions from the focus groups. Table 7 lists a summary of the suggested practices and improvements, organized by category.

4.4.1 Increasing customer involvement in acceptance testing

The practices suggested in this section are mostly related to the potential benefits discussed in section 4.3.1. That is, they should help create a shared understanding of the software between the customer and the technical team, by involving the customer more in the process of writing acceptance tests.

The most obvious way to increase customer involvement in the acceptance testing would be to have the customer define the acceptance tests, as suggested by participant A4. On a similar note, participant B1 emphasised that it is important that the team demands the customer to specify the test cases to the technical team. However, in many cases the customer is not technically skilled enough to be able to produce actual test cases. Participant B1 then elaborated that the important thing is that the customer provides acceptance criteria for each requirement; they do not have to be in the form of test cases. In addition, the acceptance criteria should be specified for each requirement immediately after the requirement has been defined. The criteria can be written down in e.g. natural language at first, and then be converted into automated test cases later.

Participant A2 suggested that in many cases, it would help to write the tests in a format that the customer can understand. Thus, the tests should be written in a natural language that the customer understands. Robot Framework – which is currently the 'go-to' tool for acceptance testing in the case company – takes a keyword-driven approach to testing, and thus enables writing the tests in a natural language. One participant noted that for some customers, the tests should be written in Finnish, as the customer is not necessarily confident in their skills in English. A2 also suggested that tests could be run with the customer present, so that the customer could have a better understanding of the tests and their status. This could be done regularly e.g. at the end of each sprint. The focus group participants also mentioned that it would help if the customer gave more information on what should be paid attention to in acceptance testing. The technical team could discuss with the customer about which parts of the software are the most important, so the testing effort could be prioritized better.

Table 7: Suggested good practices for integrating automated acceptance testing with requirements engineering, organized by category.

Category	Suggestions for future practices
Increasing customer involvement in acceptance testing	• Customer should define acceptance criteria for each requirement • Writing acceptance tests in a format that the customer can understand • Periodically running acceptance tests in the presence of the customer • Technical team should discuss with customer about which parts of the software are the most important
Using tags to link test cases and requirements	• Link acceptance test cases to tickets in requirements management system with identifying tags • Should be able to see the status of acceptance tests and requirements easily from requirements management system
Create an internal set of best practices	• More concrete set of test automation guidelines should be written for internal use at software development companies • Should write guidelines for adopting test automation also in pre-existing and legacy projects • Unify test automation practices with company-wide guidelines
Testing environments	• Ensure that testing environment is stable and easy to set up • Testing environment should be separate from other environments • Use virtualized environments for testing
Pilot project	• Software development companies should aim to do a pilot project where the linking of automated test cases and requirements is utilized

4.4.2 Using tags to link test cases and requirements

The practices suggested in this section are mostly related to the potential benefits discussed in section 4.3.2. They should help in providing a better visibility to the implementation status of requirements.

Multiple focus group participants suggested the practice of linking acceptance test

cases to tickets contained in a requirements management system - such as Jira - with tags. This would be done by attaching some kind of an identifying tag to each test case, describing which ticket the test cases are associated with. As each requirement is represented by a ticket or a set of tickets in the requirements management system, the tags would thus create links between each requirement and their associated acceptance test cases.

Simply linking the test cases to tickets with tags might help the technical team to some degree, but the real benefit from linking the two is being able to create a better picture of the status of the acceptance tests, and by extension, the status of requirements. Participants B1 and B4 stated that they would like to see directly from Jira (or any other requirements management system) the acceptance tests associated to each requirement, and their statuses. This would make information about the status of acceptance tests and requirements more easily available to both the technical team and non-technical project participants. Unfortunately, no tool for directly linking together Robot Framework test cases with Jira tickets currently exists. Thus, developing a Robot Framework test case integration for Jira should be taken under consideration at the case company. Participant B1 also suggested that visualizing information about the status of tests and requirements – e.g. in the form of a graph – would be helpful for project participants.

4.4.3 Create an internal set of best practices

The focus group participants brought up many times during the focus groups that the 'test automation good practice guidelines' listed in the test automation guide [16] were ambiguous and written on a too general level. This was also apparent during the focus groups, as there was quite a bit of uncertainty amongst the participants when they were asked if some of the good practices had been followed in the target projects. As a result, multiple focus group participants suggested that a new, more practically oriented and more concrete set of good practice guidelines for test automation should be written for internal use at software development companies.

Participant A1 mentioned that currently the company seems to have best practice guidelines for adopting test automation when starting a new project from scratch. However, no guidelines for adopting test automation in ongoing or legacy projects exist. Thus, A1 suggested that the company should also create a process for adopting test automation practices after the beginning of a project and in legacy projects.

Participants B1 and B4 also argued that currently, there is a need to unify test automation practices. A lot of test automation work has been done for external clients, with good results. However, many internally developed software projects – especially older projects that are currently in support phase – lack the resources and practices to properly adopt test automation. The focus group participants felt that there's a lot of test automation know-how in the company, but that it is spread unevenly. Participant B1 suggested that a set of company-level test automation practices would help unify how test automation is utilized in the company. It would also help in spreading the test automation know-how to more people in the company.

4.4.4 Testing environments

The potential problems that testing environments can cause to the automated testing effort are explored more thoroughly in section 4.5.4; this section focuses primarily on presenting improvement suggestions for testing environments. Multiple participants from the second focus group mentioned having had problems with testing environments during the project. As a result, they also had some suggestions for improving things with regards to testing environments.

Participant B3 mentioned that setting up the testing environment in project 2 had been complicated and took a lot of effort. Participant B4 added that the testing environment had also been rather unstable, as the environment had a tendency to crash occasionally. Unstable or cumbersome testing environments easily become a bottleneck in the testing effort, preventing the technical team from receiving the benefits that automated testing would otherwise provide. Thus, the focus group participants emphasised that care should be taken to ensure that the testing environment is stable and reliable.

Participant B5 suggested that the environment where acceptance tests are run should be separate from other environments. This would enable the use of consistent data for each test execution. He also suggested that using virtualized environments – such as Vagrant boxes – would help in fulfilling the need for separate testing environments. However, participant B5 also emphasised that the process for setting up the virtualized environments should be as streamlined as possible, or otherwise the virtualized environments could end up hindering the testing effort instead of helping it.

4.4.5 Pilot project

Participant B1 suggested that software development companies should aim to do an internal pilot project, in which the linking of automated test cases and requirements would be utilized, and this practice would also be taken into account in all aspects of the project, including sales. A pilot project would help to verify which of the potential benefits discussed in section 4.3 could actually be achieved in practice. It could also help convince customers of the benefits of linking automated tests and requirements, by producing data about how well it works in practice. Participant B1 especially argued that potential customers usually believe in data as a sales argument, and a pilot project would be a good opportunity to produce such data.

4.5 RQ2 - What kind of challenges might be encountered when integrating automated acceptance testing with requirements engineering?

Based on the discussions from both focus groups, four categories of challenges that might hinder the integration of automated acceptance testing and requirements engineering were identified. Table 8 lists a summary of these challenges, organized by category.

Table 8: Challenges that hinder the integration of automated acceptance tests with requirements engineering when encountered, organized by category.

Category	Challenges
Adopting test automation in pre-existing projects	• Hard to write new tests if 'foundation' on which to build on is missing • Investment required for overcoming accumulated technical debt in test automation might not be reasonable • More laborous to write test cases if testability not considered already during development
Lack of processes for test automation	• Test automation practices lack cohesion • Lack of practical guidelines for utilizing test automation
Convincing customers	• Hard to convince customers to invest more into test automation • Not enough data available that could be used to demonstrate benefits of test automation to customers
Problems with testing environments	• Testing environments provided by customers often contain additional constraints or issues • Unstable testing environment often means that test results cannot be trusted

4.5.1 Adopting test automation in pre-existing projects

As already mentioned in section 4.4.3, many problems hinder the adopting of test automation in pre-existing projects and in projects that are in support phase of their lifecycle. These issues were mostly brought up by participants in the first focus group, as project 1 was already in the support phase at the time of the interview, while project 2 was just nearing completion.

Participant A1 explained the difficulties of writing tests into pre-existing projects. In many support phase projects, the 'foundation' on which to build tests is lacking. That is, technical debt related to testing has been accumulating since the beginning of the project, and it has gotten worse the longer the project has been going on. In order to start writing automated tests to such a project, one would have to first spend a lot of effort on bringing the whole project's testing foundation up to speed. This includes e.g. generating test data and writing tests for at least part of the already existing features. Updating the whole test foundation is a large undertaking, and this

translates to it costing a large amount of money. The investment for bringing the tests up-to-date after the software has already been deployed to production might not be reasonable for the customer.

The situation is especially problematic in projects where automated testing has not been utilized during development, and where the software has been in production use for many years. At this point, overcoming the technical debt in testing would be so effort-intensive – and thus, expensive – that it is unlikely that the customer can be convinced to pay for adopting automated testing. This means that legacy software that is still being actively supported is often unable to adopt automated testing during support phase, as the investment for implementing test automation might not be reasonable.

Participant A3 stated that usually in projects where testability has not been taken into consideration during development, writing new test cases is slower than it would otherwise be. For example, the user interface elements might lack an ID that would make selecting that element easy when writing test cases. Instead, one would have to find the element by writing e.g. a CSS or XPath statement, which selects that particular element. This of course causes some extra work when writing the tests. This and many similar technical details, when combined together, can significantly hinder the writing of new test cases. Most of these issues are noticed and will be easily fixable if automated test cases are written alongside development. However, for a person who has not been involved in writing the original code, fixing such issues at a later time can prove to be very laborous.

4.5.2 Lack of processes for test automation

As discussed previously in section 4.4.3, the focus group participants felt that test automation practices currently seem to lack cohesion on a company level. Participant A1 mentioned that there usually are no general guidelines for adopting test automation in projects, or how to account for test automation in effort estimations and in sales. That is, there's still quite a bit of uncertainty on how test automation should be utilized in software development projects, and how can it be ensured that test automation is used in a way that actually provides value for the customer.

As for the practice of linking automated acceptance tests and requirements management, participant B5 argued that the fact that there is no existing pilot project on the company level limits its potential adoption in projects. Before the practice could be adopted more widely, a pilot project showing the benefits for the customer should be done.

4.5.3 Convincing customers

One of the most critical challenges in utilizing the practice of linking automated acceptance tests with requirements management (and test automation in general) is the aspect of selling the idea to the customers. Participant B1 stated very plainly that the customers are not intrinsically interested in the technical aspects of the practice. Some possible benefits of the practice, such as providing better visibility into the requirements' implementation status (as discussed in section 4.3.2), do

provide some additional value by themselves to non-technical customer personnel. However, participant B1 argued that the primary topic of interest from customers' point of view is how utilizing this practice can save them money in the long run. As a general rule, the most important factor for convincing customers of the benefits of any practice in a software development project is the potential to save money.

In order to work properly, the practice of linking automated acceptance tests with requirements management requires that the technical team writes and maintains a decent set of automated acceptance tests. Thus, the practice requires a monetary investment into the testing effort. In general, utilizing test automation costs more money than developing software without it in the short term; the benefits are usually only seen in the longer term. From the customer's point of view, the value proposition is the following: invest more money into the development work in the short term, and the overall cost of the development work will be smaller in the long term. However, the focus group participants felt that it is hard to convince the customer that the additional investment in test automation in the short term will pay off. Having good, concrete data on which to base the argument would often go a long way towards convincing the customer. As mentioned before in section 4.4.5, participant B1 suggested that doing a company-level pilot project could help provide data which would demonstrate that the investment in test automation has paid off in a previous project. Additionally, participant B1 commented that selling the customer on the idea that the development team will also have a person that will focus solely on writing automated tests is much easier when doing an agile project when compared to fixed-price projects.

4.5.4 Problems with testing environments

Most of the issues mentioned brought up in this section were mentioned by participants in the second focus group. This is mostly because in project 2, more effort was invested in to the testing effort, and thus the project team encountered more practical issues related to the testing environments compared to project 1.

Participant B1 mentioned that when doing agile development, the customer might want the development team to work on-site in the customer's premises. This could also mean that the testing environments available to the development team are provided by the customer; this was also the case in project 2. Participant B1 also felt that environments provided by the customer can have at least some additional constraints or issues that the environments wouldn't have had if they'd been provided by the development team's company.

In the case of project 2, the most significant problem with the testing environment was its instability. The environment occasionally crashed completely, preventing the development team all access there. There were also problems with network connection timeouts in the testing environment, which occasionally caused the automated test runs to fail. Participant B5 said that these problems made testing very frustrating at times, as a failing execution of the automated tests was not necessarily caused by a failure in the software. These false alarms meant that the developers could not trust the results of the test runs. This meant that the information value of the test

runs quickly diminished, as a failing test run might or might not mean that there was something wrong with the code changes the developers had done. Automated testing seems to be very sensitive to problems in the testing environments, and the benefits are easily diminished if there are problems with the testing environment.

5 Discussion

5.1 RQ1 - Most important benefits

One of the most important benefits that could possibly be acquired by linking automated acceptance tests to requirements engineering is the aspect of providing a shared understanding of the software between the customer and the technical team. A shared understanding of the requirements and the value that the developed software provides to the customer should help reduce errors in both the definition and implementation of requirements. Thus, it should ensure that no progress on the software has to be discarded due to misunderstood requirements, and that the completed software fulfills the customer's needs.

The ability to provide more accurate information about the status of requirements is another important benefit that could be acquired by linking automated acceptance tests to requirements engineering. By linking together acceptance test cases with tickets in a requirements management system, one can create a more accurate picture of the status of acceptance tests and the requirements that they are associated with. The key factor here is easiness: by making status information of tests and requirements available in an easily understandable format, both technical and non-technical project participants are able to have a better understanding on how the implementation of the requirements is progressing.

5.2 RQ2 - Challenges of adopting

One of the most significant challenges that currently hinders the practice of integrating automated acceptance testing with requirements engineering is selling customers on the idea. The linking of automated acceptance tests with requirements necessitates that a decent set of automated acceptance test cases for the software is written and maintained by the technical team. This means that the customer must invest more money than they otherwise would into the testing effort at the start of the project. This is an investment that customers are usually unwilling to make without being presented with enough data that proves the benefits of the practice; unfortunately, such data is currently hard to find.

Lack of cohesion in test automation processes in software development companies also seems to be a major challenge. There might be significant test automation know-how in the company, but it is usually spread unevenly between employees. The focus group participants stated that there is uncertainty about how test automation should be utilized in software development projects. They called for more practical guidelines on test automation practices that would unify test automation practices on a company level.

5.3 RQ3 - Suggested good practices

The focus group results indicate that the customer should be more involved in defining the acceptance tests. It is important that the customer – possibly with help from

the technical team – defines acceptance criteria for each requirement as soon as a requirement has been defined. Additionally, the technical team should discuss with the customer which parts of the software are the most important, so that testing effort can prioritized accordingly. Both of these practices aim to help in creating a shared understanding of the software between the customer and the technical team.

The focus group results also strongly suggest using the practice of linking automated acceptance test cases to tickets in a requirements management system with identifying tags. This practice could be enhanced further by showing the status information of acceptance tests and their associated requirements directly in the requirements management system. This would provide the status information in a more easily understandable format, allowing for a better picture of the software's implementation status for both technical and non-technical project participants.

Another important finding was that there is a need for an internal set of 'best practices' related to test automation. These best practice guidelines should be practically-oriented, low-level guidelines on how to utilize test automation in software development projects. They should also cover the scenario where test automation is adopted at a later stage in the project's lifecycle. The guidelines should then be adopted company-wide in order to unify the use of test automation practices in the company.

Lastly, the focus group participants suggested that software development companies should do an internal pilot project on the linking of automated test cases and requirements. The practice should be taken into account in all aspects of the project, including sales. A pilot project should help provide data on how well the practice works in real software development projects. This data could also be used as rationale for convincing customers on the benefits of the practice.

6 Conclusions

The research problem this thesis aimed to answer was: how can automated acceptance testing be integrated with requirements engineering in a beneficial manner? In other words, how should the integration of automated acceptance testing with requirements engineering be handled, so that integrating these two areas of software development provides some tangible benefits?

Results of the empirical study suggest that one of the most important potentially achievable benefits of this integration is being able to provide more accurate information about the status of each requirement's implementation. Based on the results, the practice of tag-based linking between automated acceptance test cases and tickets in a requirements management system would help in achieving this benefit. However, no tools for easy utilization of tag-based linking currently exist. In order to be able to further explore the use of this practice, such a tool should be developed. An example of a straightforward approach for this would be to create a piece of software that parses results from Robot Framework test reports, and shows the result information in Jira, under the ticket that is associated with the tests.

Another important factor in achieving the possible benefits of integrating automated acceptance tests with requirements engineering is increasing the customer's involvement in the process of defining acceptance tests. The customer's involvement in the process should help in creating a shared understanding of the software that is being developed between the customer and the technical team. The whole practice of linking acceptance tests to requirements suffers greatly if the acceptance tests do not accurately depict the software's expected behavior. If there is a major disconnect between the acceptance tests and the customer's expectations on the software's behavior, a set of passing acceptance tests for a feature does not mean that the feature is actually completed. In this situation, the status of acceptance tests no longer reflects the status of requirements, and the benefits of the practice are lost. Thus, the customer's involvement in the process is extremely important to ensure that the tag-based linking of acceptance tests and requirements provides the expected benefits.

Future studies on the topic should address the issue of there not being enough data that demonstrates the viability and benefits of integrating automated acceptance tests and requirements engineering in practical software development work. More information on the topic could be gained by performing multiple case studies, where the recommended practices for integrating automated acceptance testing with requirements engineering are utilized in a software development project. This information could help to ensure that the most important benefits of these practices are able to be realized consistently in real software development projects, and to gain more insight into potential problems in adopting the practices.

References

[1] David Talby, Ori Nakar, Noam Shmueli, Eli Margolin, and Arie Keren. A process-complete automatic acceptance testing framework. In *Software-Science, Technology and Engineering, 2005. Proceedings. IEEE International Conference on*, pages 129–138. IEEE, 2005.

[2] Paul Ammann and Jeff Offutt. *Introduction to software testing*. Cambridge University Press, 2008.

[3] Elizabeth Bjarnason, Per Runeson, Markus Borg, Michael Unterkalmsteiner, Emelie Engström, Björn Regnell, Giedre Sabaliauskaite, Annabella Loconsole, Tony Gorschek, and Robert Feldt. Challenges and practices in aligning requirements with verification and validation: a case study of six companies. *Empirical Software Engineering*, 19(6):1809–1855, 2014.

[4] Eero J Uusitalo, Marko Komssi, Marjo Kauppinen, and Alan M Davis. Linking requirements and testing in practice. In *International Requirements Engineering, 2008. RE'08. 16th IEEE*, pages 265–270. IEEE, 2008.

[5] Jyrki Kontio, Laura Lehtola, and Johanna Bragge. Using the focus group method in software engineering: obtaining practitioner and user experiences. In *Empirical Software Engineering, 2004. ISESE'04. Proceedings. 2004 International Symposium on*, pages 271–280. IEEE, 2004.

[6] Maya Daneva. Focus group: Cost-effective and methodologically sound ways to get practitioners involved in your empirical re research. In *REFSQ Workshops*, pages 211–216, 2015.

[7] Richard A Krueger and Mary Anne Casey. *Focus groups: A practical guide for applied research*. Sage publications, 2014.

[8] Pierre Bourque, Richard E Fairley, et al. *Guide to the software engineering body of knowledge (SWEBOK (R)): Version 3.0*. IEEE Computer Society Press, 2014.

[9] Glenford J Myers, Corey Sandler, and Tom Badgett. *The art of software testing*. John Wiley & Sons, 2011.

[10] Borge Haugset and Geir Kjetil Hanssen. Automated acceptance testing: A literature review and an industrial case study. In *Agile, 2008. AGILE'08. Conference*, pages 27–38. IEEE, 2008.

[11] Børge Haugset and Geir K Hanssen. The home ground of automated acceptance testing: Mature use of fitnesse. In *Agile Conference (AGILE), 2011*, pages 97–106. IEEE, 2011.

[12] Marja Käpyaho and Marjo Kauppinen. Agile requirements engineering with prototyping: A case study. In *2015 IEEE 23rd International Requirements Engineering Conference (RE)*, pages 334–343. IEEE, 2015.

[13] Børge Haugset and Tor Stålhane. Automated acceptance testing as an agile requirements engineering practice. In *System Science (HICSS), 2012 45th Hawaii International Conference on*, pages 5289–5298. IEEE, 2012.

[14] Elizabeth Bjarnason, Krzysztof Wnuk, and Björn Regnell. A case study on benefits and side-effects of agile practices in large-scale requirements engineering. In *Proceedings of the 1st Workshop on Agile Requirements Engineering*, page 3. ACM, 2011.

[15] Eero Uusitalo. *Software requirements engineering and acceptance testing as cooperating disciplines*. 2009.

[16] Marko Klemetti and Tatu Kairi. *Testiautomaatio ja Robot Framework -opas*. 2016.

A Appendix A: Script for focus group interviews (in Finnish)

- Opening n. 10min(?)
  - Pohjustus: ”Mikä on tämän fokusryhmän tarkoitus”
    - ”Tarkoituksena on saada lisää ymmärrystä siihen, miten voidaan liittää automaattinen hyväksyntätestaus vaatimusten määrittelyyn ja hallintaan”
  - Käydään läpi käytännön järjestelyt, miten fokusryhmä-tilaisuus etenee
  - ”Kuvaa lyhyesti oma roolisi projektissa”
    - jokainen osallistuja kertoo oman roolinsa
- Introduction & Transition n. 45 min(?)
  - Pääteema: ”Automaattisen hyväksyntätestauksen hyvät käytännöt, nykytilanne”
    - ”Mitä testiautomaatiokäytäntöjä projektissa käytettiin sinun mielestäsi”
      - Osallistujat kirjoittavat post-it -lapuille ylös lyhyen listauksen projektissa käytetyistä testiautomaatiokäytännöistä
      - saattaa vaatia vähän jotain probea, esim. antaa esimerkin jostain hyvästä testiautomaatiokäytännöstä, esim. ”Automaattitestit pidettiin jatkuvasti ajan tasalla projektin ajan”
    - Käydään laput läpi yhdessä, keskustelua
      - tässä kohti saadaan käytettyjä hyviä käytäntöjä lapuille
    - NÄYTÄ: numeroitu listaus Eficoden testiautomaatio-oppaan hyvistä testiautomaatiokäytännöistä (ne neljä dippa-aiheen kannalta relevanttia) yksitellen
      - äänestys post-it -lapuille, että onko ko. käytäntöä käytetty projektissa
    - ”Miten testitapaukset kirjoitettiin projektissa?”
      - ”Ketkä olivat mukana kirjoittamassa testejä?”
      - ”Missä vaiheessa projektia hyväksyntätestien kirjoittaminen aloitettiin”
  - Pääteema: ”Vaatimustenhallinta ja asiakkaan rooli”
    - ”Miten vaatimuksia hallittiin projektissa?”
    - ”Miten asiakas on osallistunut tähän projektiin?”
- Key n. 45 min(?)
  - Pääteema: ”Automaattisten hyväksyntätestien liittäminen vaatimusten määrittelyyn ja hallintaan”

  - Aliteema: RQ3: ”Miten se tehdään”
    * ”Miten automaattisella hyväksyntätestauksella voitaisiin paremmin tukea vaatimusten määrittelyä ja hallintaa?”
      · NÄYTÄ: kaaviokuva asiasta
      · ”Mitä työkaluja tai käytäntöjä kannattaisi ottaa käyttöön tai vaihtaa toisiin?”: osallistujat kirjoittavat ideoitaan post-it -lapuille -> Ideoita uusista hyvistä käytännöistä?
  - Aliteema: RQ1: ”Miksi tälläista kannattaa tehdä”
    * ”Mitä hyötyä voitaisiin saada automaattisten hyväksyntätestien yhdistämisestä vaatimusten määrittelyyn ja hallintaan?”
      · Tavallinen haastattelukysymys
  - Aliteema: RQ2: ”Mitä hankaluuksia/ongelmia tässä voi tulla”
    * ”Mitä ongelmia voi tulla automaattisten hyväksyntätestien yhdistämisestä vaatimusten määrittelyyn ja hallintaan?”
      · Tavallinen haastattelukysymys

- Ending n. 5min
  - ”Olisiko vielä jotain mitä haluaisitte nostaa esiin?”
    * liittyen joko:
      · Automaattiseen hyväksyntätestaukseen
      · Vaatimustenhallintaan
      · tai niiden integrointiin?
      · tämä on samalla myös yhteenveto, että näistä asioista keskustelimme
  - Kiitä tiedoista
  - Miten aion jatkaa näiden tietojen kanssa
    * Analysoidaan ja kirjoitetaan dippaan
    * Mahdollisuus myös siihen, että kysytään vielä lisää tarkennusta esim. Slackissa